DReCon: Data-Driven Responsive Control of Physics-Based Characters

KEVIN BERGAMIN, McGill University, Canada
SIMON CLAVET, Ubisoft La Forge, Canada
DANIEL HOLDEN, Ubisoft La Forge, Canada
JAMES RICHARD FORBES, McGill University, Canada

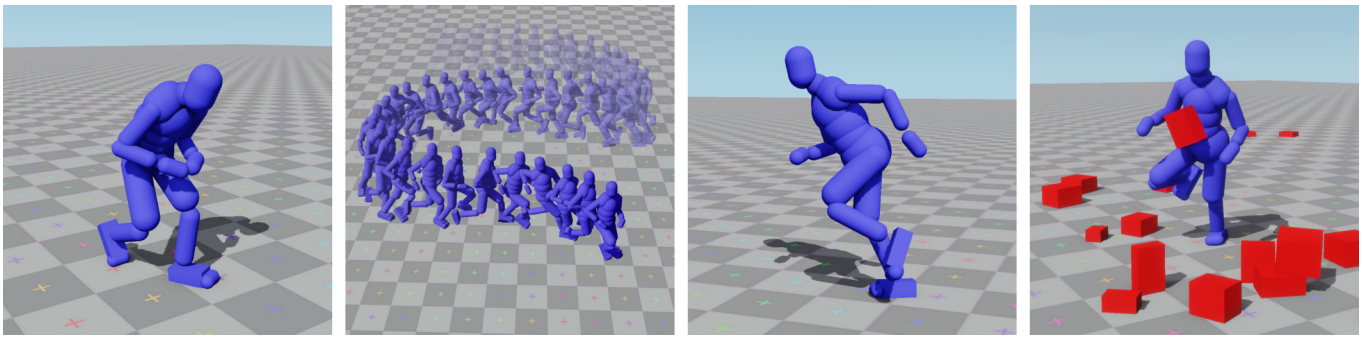

Fig. 1. A physically simulated character demonstrating user-controlled motion in real-time. The motion is interactively generated based on the content of a 10 minute motion database, in order to responsively meet user input requirements while maintaining a natural style and high robustness to perturbations.

Interactive control of self-balancing, physically simulated humanoids is a long standing problem in the field of real-time character animation. While physical simulation guarantees realistic interactions in the virtual world, simulated characters can appear unnatural if they perform unusual movements in order to maintain balance. Therefore, obtaining a high level of responsiveness to user control, runtime performance, and diversity has often been overlooked in exchange for motion quality. Recent work in the field of deep reinforcement learning has shown that training physically simulated characters to follow motion capture clips can yield high quality tracking results. We propose a two-step approach for building responsive simulated character controllers from unstructured motion capture data. First, meaningful features from the data such as movement direction, heading direction, speed, and locomotion style, are interactively specified and drive a kinematic character controller implemented using motion matching. Second, reinforcement learning is used to train a simulated character controller that is general enough to track the entire distribution of motion that can be generated by the kinematic controller. Our design emphasizes responsiveness to user input, visual quality, and low runtime cost for application in video-games.

CCS Concepts: • **Computing methodologies** → **Animation**; *Control methods*; *Reinforcement learning*; *Physical simulation*;

Additional Key Words and Phrases: physically based animation, reinforcement learning, motion capture, real-time graphics

ACM Reference Format:
Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. DReCon: Data-Driven Responsive Control of Physics-Based Characters. *ACM Trans. Graph.* 38, 6, Article 206 (November 2019), 11 pages. https://doi.org/10.1145/3355089.3356536

Authors' addresses: Kevin Bergamin, McGill University, 845 Sherbrooke Street West, Montreal, QC, H3A 0G4, Canada, kevin.bergamin@mail.mcgill.ca; Simon Clavet, Ubisoft La Forge, 5505 St Laurent Blvd, Montreal, QC, H2T 1S6, Canada, simon.clavet@ubisoft.com; Daniel Holden, Ubisoft La Forge, 5505 St Laurent Blvd, Montreal, QC, H2T 1S6, Canada, daniel.holden@ubisoft.com; James Richard Forbes, McGill University, 845 Sherbrooke Street West, Montreal, QC, H3A 0G4, Canada, james.richard.forbes@mcgill.ca.

0730-0301/2019/11-ART206 $15.00
https://doi.org/10.1145/3355089.3356536

1 INTRODUCTION

In many interactive applications, such as video-games, physical simulation has been adopted for the modelling of a number of complex phenomenon including dynamics of cloth, fluids, and rigid bodies. However, physical simulation has seen little adoption for use in character animation, and is most often applied in limited scope such as for secondary motion, or to produce passive ragdolls that are enabled only for uncontrolled tumbling.

Historically, the core challenge of implementing physically based characters has been in producing a self-balancing controller that is robust, can produce realistic movements, and is adaptable enough to maintain control in all possible situations that may be presented. Recently, research with reinforcement learning (RL) based methods has shown promise in overcoming some of these problems, and it is now possible to design physically based character controllers that can effectively follow motion capture data with a high degree of quality [Liu and Hodgins 2018; Peng et al. 2018]. Yet, such controllers lack responsiveness when steered by user input, cannot scale easily to a large diverse set of motions, and can be prohibitively expensive, making them unsuitable for the kind of real-time character control required in video-games.

In this paper, we propose a new approach to physically based character control that allows for a high degree of responsiveness, while retaining the natural visual qualities of human motion. Our

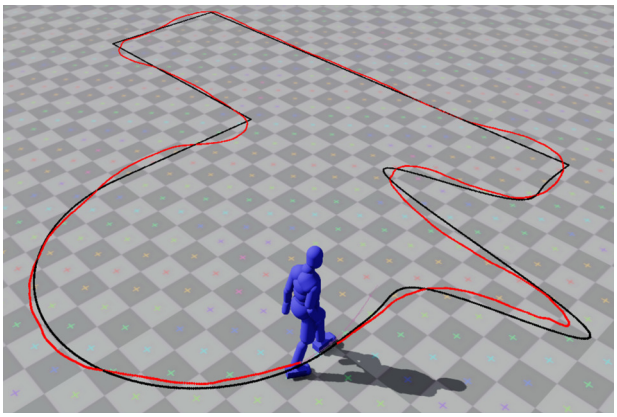

Fig. 2. A path with various turns (black line) can be followed by a user-controlled physically simulated character with high accuracy. Red line shows the path traveled by the character's projected center of mass. Floor checkerboard tiles are 0.5m × 0.5m. Walking pace was maintained throughout.

method consists of two key steps. We use an unstructured motion capture database to synthesize natural looking kinematic human motion that follows user specified control changes in real-time. At a regular interval, parameters such as the desired speed and facing direction are provided to a kinematic controller implemented using motion matching [Clavet 2016]. This kinematic controller uses features extracted from the animations and user input to continually find the best animation segments to play. This performs short-horizon planning in kinematic space, but retains the intent of achieving high-level user control goals. The joint angles from the kinematic motion then provide open-loop control of a simulated character as joint-level proportional-derivative (PD) control targets. This open-loop control is not sufficient for maintained character balance, but comes close. Reinforcement learning is used to find a suitable feedback policy to combine with open-loop control of the simulated character. The policy is capable of maintaining balance and powerful enough to generalize across any possible target motion that is produced by the kinematic controller. This feedback policy provides corrective changes to the PD control targets for a subset of the joints, which allows the user controlled target motion to be followed by the physical character without falling. The result is a state-of-the-art controller for physically simulated humanoids that can responsively change direction, heading, and style, while retaining visual quality, balance and robustly correcting for disturbances. The controller has minimal runtime cost, easily enabling integration in current generation interactive applications.

2 RELATED WORK

We review previous research from the domains of physically based animation, robotics, and control, with a specific focus on humanoid locomotion. Geijtenbeek and Pronost [2012] gives a general introduction to this research area.

Manual design: A variety of work focuses on designing task specific heuristics which achieve character control. Yin et al. [2007] utilize a pose-based finite state machine to maintain balance and follow motion capture reference. Lee et al. [2010] modulate and transition between motion capture reference data in combination with heuristic based balance control. A large body of work employs heuristic driven carefully designed methods [Brown et al. 2013; Coros et al. 2010; Geijtenbeek et al. 2013; Raibert and Hodgins 1991], but all usually require significant manual tuning, and produce controllers that, while robust, are difficult to control stylistically. Some methods [da Silva et al. 2017; Zordan and Hodgins 2002] have been developed which use external forces for added stability, allowing for more stylistic control at the cost of physical realism.

Online optimization: Control actions can be found each step which maximize some performance goal, but this generally requires expensive optimization procedures. Model predictive control, employing optimization over short predictive horizons and/or using approximate models, has been applied to locomotion [da Silva et al. 2008; Tassa et al. 2012] with impressive robustness. Machietto et al. [2009] apply a similar approach to control angular and linear momentum, as well as center of pressure, but are limited to balance related tasks with no stepping. Jain and Liu [2011] apply long horizon planning on a subset of modal coordinates every time step to find actions which track motion capture reference.

Offline optimization: Methods which optimize trajectories and feedback strategies offline in order to meet performance goals have been extensively developed. Liu et al. [2015; 2010] optimize open-loop controllers to follow motion capture reference clips, but these are generally sensitive to initial conditions and disturbances. Simple feedback strategies have also been optimized offline with success [Ding et al. 2015; Lee et al. 2014; Muico et al. 2009], and some methods have been explored which transition between optimized behaviours [Liu and Hodgins 2017; Liu et al. 2016; Sok et al. 2007].

Reinforcement learning: There has been a recent surge in work using deep RL (DRL) to train motor control policies for simulated characters [Chentanez et al. 2018; Heess et al. 2017; Liang et al. 2018; Liu and Hodgins 2017, 2018; Peng et al. 2018, 2015, 2016, 2017]. These methods find effective control policies by trial-and-error experience gathering and offline optimization. Experience is utilized to optimize a control strategy such that it maximizes long term sums of rewards. Rewards are designed to encourage high-level behaviors such as forward movement, balance, and reference motion tracking. For problem domains with continuous state and action spaces, a parameterized stochastic control policy, usually a deep neural network, can be optimized directly using the policy gradient method [Sutton and Barto 2018]. The control policies take as input the current state and output a distribution of optimal actions. A deterministic policy derived from the stochastic policy can be used outside training where maximum performance and determinism is desirable. Once trained, DRL methods often produce cheap to evaluate, performant, and robust controllers. This is because policies are trained to take actions that are optimal over long time periods, and DRL methods generalize to a wide variety of problem domains [Brockman et al. 2016].

Heess et al. [2017] design a variety of environments and simple rewards to encourage learning of robust locomotion policies based on high-level task requirements which increase in difficulty during training. While successful, the aforementioned work shows a tendency towards unnatural motion styles. A solution to this is to offer

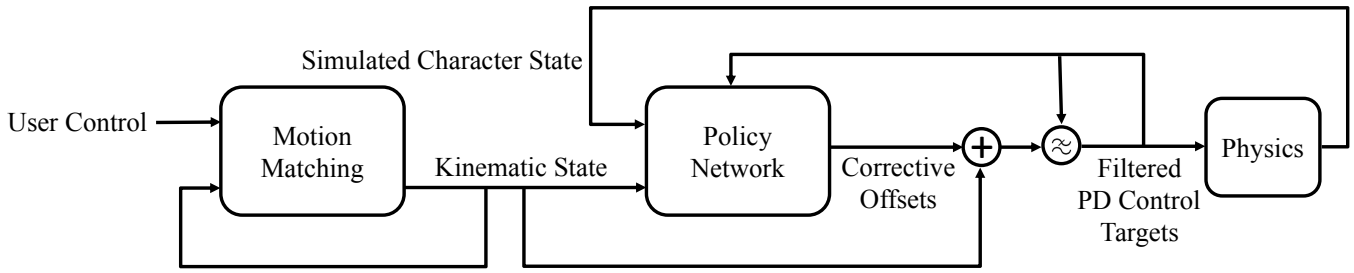

Fig. 3. Our controller starts by generating a stream of kinematic animation that satisfies user control. Every 10 frames the motion matching system finds a small segment of motion capture data that optimally matches the current pose and the desired future velocity. In a second step, a balance feedback policy adds corrective offsets to PD-control targets, that are finally filtered and sent to the physics engine.

an explicit reward for tracking a reference motion clip, a strategy used in recent work with great success [Chentanez et al. 2018; Peng et al. 2018, 2017].

In Peng et al. [2018] (DeepMimic) simulated characters are rewarded for tracking poses from short motion capture clips, as well as for achieving hardcoded per-controller secondary goals. Robust control policies that can survive perturbations while retaining similar style to the motion capture clips are learned. The resulting motion of the physical character is difficult to distinguish from the clip it has been trained to imitate. Only a few short clips are learned per controller, or controllers are combined, each requiring significant training. A scalar phase variable encodes animation progress, limiting policies to tracking a few (aligned) animations.

Chentanez et al. [2018] build upon DeepMimic's approach, training a single policy on a large database of animation. Policies are trained to apply torques on top of a simple PD controller driven by the animation. This work provides the policy with the desired animation pose, eliminating use of phase and helping with generalization. This demonstrates the possibility of generalizing on a large database at the cost of reduced quality.

Concurrent to our work, Park and Lee [2019] presents a method that trains an RNN based motion generator on unorganized motion capture, and utilize DRL to train a controller to track the generated motion.

In this paper we take inspiration from recent work, and build a DRL controller that can learn from a large database of unstructured motion data, that is capable of generating high quality motion with a high degree of responsiveness to user input, all the while optimizing the design towards low runtime cost.

3 OVERVIEW

Our method consists of the following stages. First, we acquire a database of skeletal motion capture consisting of 10 minutes of unstructured locomotion data. This data includes walking, running, and crouching. Next, we skin a character mesh to this skeleton, and use an automated capsule and box fitting procedure to build a physical model of the character that matches the mesh in terms of surface and volume (see Fig 4). We proceed to extract a selection of features from the motion capture data and use these features to build a database suitable for a kinematic controller implemented using motion matching (see Section 5).

Using the kinematic controller and physics simulation, we build a virtual environment. During training, we emulate a user pressing buttons and moving the sticks on a gamepad, generating many different kinematic trajectories from the kinematic controller that cover a wide distribution of behaviours and transitions. Using this virtual environment and artificial user, we train a feedback policy that outputs per joint control adjustments for the simulated character (see Section 6). This policy is trained using a reward signal that includes terms for not falling over, for remaining close to the kinematic target motion, and moving in a manner consistent with user input. Once trained, a human user can control the simulated character using a gamepad. For a schematic of our controller design please see Fig 3.

4 DATA ACQUISITION

In this section we describe our process of data aquisition, and construction of the physical model used for our simulated character.

4.1 Motion Database

Our motion database comes from a high quality multi-camera marker based motion capture system running at 60 frames/second [Vicon 2018]. The capture session itself consists of "unstructured" motion capture where by an actor walks, runs, strafes, and crouches in a variety of directions at various speeds. Around 10 minutes of data is gathered, covering a large, continuous distribution of movement, turning, pivoting, and style transitions. Different gaits such as standing, walking, running, and crouching are labelled in the data manually by selecting frame ranges (although this task is made easy by the long structure of the takes where the actor is encouraged to maintain the same gait).

4.2 Physical Model

Our character's rigid body structure is generated automatically from the motion capture skeleton using an associated human mesh. For each bone we use CMA [Hansen 2006] to best fit capsule geometries (boxes in the case of the feet) to the mesh vertices that are most rigidly skinned to each bone. This ensures that the rigid body geometries have similar surfaces and volumes to the mesh. When

Table 1. Comparison of the physical character definition used in our work with related work. **this value was not explicitly reported in Chentanez et al. [2018], rigid bodies were manually counted from video footage.*

Method	Rigid Bodies	Degrees of Freedom
(Ours)	23	54
Peng et al. [2018]	13	34
Chentanez et al. [2018]	15*	29

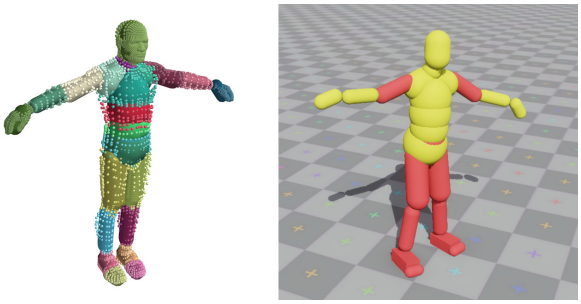

Fig. 4. **Left:** Physical character model defined by optimizing rigid body geometry (colored volumes) to match a human character mesh (colored points) based on the motion capture actor. **Right:** Feedback policy only provides actions that alter actuation of a subset of bodies (red). Other bodies (yellow) are driven purely by open-loop control.

all the geometries have been optimized, the bones and their associated capsules are placed in the character T-pose. Many capsules intersect to varying degrees, hence we discretize the capsule based character using marching cubes and divide density equally among volume elements that intersect multiple capsules. A fixed density of 985 kg/m^3 (the average human body density) is used. The resulting character, while still imperfect, provides more realistic mass distributions for each bone, and has relatively constant density when accounting for intersections, compared to a character constructed by hand. The total mass of the character is 90 kg. Compared to other work, our character definition has almost twice the number of degrees of freedom and rigid bodies, see Table 1. Our character is of the full complexity typically seen in modern motion capture pipelines [Holden 2018], only omitting fingers. This allows most details of the character motion to be reproduced physically.

5 MOTION MATCHING

5.1 Overview

Motion matching [Clavet 2016] is a simple method for constructing kinematic controllers from unstructured motion capture data, which is arguably the state-of-the-art in production game animation [Buttner 2019; Harrower 2018; Zinno 2019]. The core idea is to regularly search an animation database for a frame that best matches both current features of the character, and desired control features. Normally these features include the current local (character space) position and global velocity of several joints, as well as future trajectory positions and orientations. If the search finds a frame that matches more closely the desired features than the currently playing animation, then a transition and blend are inserted, and the animation from the matched frame onward is played. Given that the motion databases involved can be large, and the performance constraints of video-games quite tight, an important optimization to this system is to precompute the features that are matched and to store them in a separate database where the search can be performed efficiently.

We opt to use motion matching over other kinematic controllers such as PFNN [Holden et al. 2017] since it is fast and easy to tune. Input data / parameterization can be modified in real time, with minimal manual labor and no training required.

5.2 Implementation

We now describe our implementation of motion matching in more detail. First, in an offline process, for each frame i in the motion database we compute the features that are going to be matched by the search. In our case we compute the left and right foot local positions and global velocities $\mathbf{b}_i^{lfoot} \in \mathbb{R}^3$, $\mathbf{b}_i^{rfoot} \in \mathbb{R}^3$, $\dot{\mathbf{b}}_i^{lfoot} \in \mathbb{R}^3$, $\dot{\mathbf{b}}_i^{rfoot} \in \mathbb{R}^3$, the hip global velocity $\dot{\mathbf{b}}_i^{hip} \in \mathbb{R}^3$, and trajectory positions and orientations located at 20, 40, and 60 frames in the future which are projected onto the ground plane $\mathbf{t}_i \in \mathbb{R}^6$, $\mathbf{d}_i \in \mathbb{R}^6$ (concatenating 3 xy pairs $\Rightarrow \mathbb{R}^6$). This is similar to the set of features described in Clavet [2016]. We compute the mean and standard deviation for each feature across every frame and then normalize each feature independently by subtracting its mean and dividing by its standard deviation. Normalization helps to balance cost weighting for each feature, and is not mentioned by Clavet [2016]. For each frame we then concatenate these normalized feature values into a single large feature vector $\mathbf{m}_i = \{\, \mathbf{b}_i^{lfoot}\ \mathbf{b}_i^{rfoot}\ \dot{\mathbf{b}}_i^{lfoot}\ \dot{\mathbf{b}}_i^{rfoot}\ \dot{\mathbf{b}}_i^{hip}\ \mathbf{t}_i\ \mathbf{d}_i \,\} \in \mathbb{R}^{27}$ and concatenate these feature vectors into a matrix to form our matching database $\mathbf{M} = [\mathbf{m}_0, \mathbf{m}_1, ..., \mathbf{m}_{n-1}] \in \mathbb{R}^{N_{frames} \times 27}$, where N_{frames} is the number of frames in the database and 27 is the total size of the feature vector.

At runtime: every 10 frames, or when the user input changes quickly, we construct a query vector $\hat{\mathbf{m}}$ that is to be compared to the frames in the database to decide if a transition is required. This is in contrast to Clavet [2016] which suggests matching every frame, which was found to be unnecessarily frequent. To construct this feature vector, we take the joint positions and velocities from the current best matching animation being played, and compute the future trajectory points by extrapolating the user input with a critically damped spring damper, which essentially mixes the current character velocity with the desired user velocity [Kermse 2004]. Given the constructed query feature vector $\hat{\mathbf{m}}$, which is normalized using the same values found during pre-processing, we then must find the frame index j that minimizes the squared euclidean distance between the query feature vector and feature vector of the frame at that particular index,

$$j = \arg\min_i \|\hat{\mathbf{m}} - \mathbf{m}_i\|^2 \tag{1}$$

After this index j has been found, if it is different to the current frame being played, we transition to the animation starting at frame j and insert a small blend to ensure a seamless transition. In our implementation we use inertialization blending [Bollo 2016, 2017] to produce this seamless transition, giving the added benefit of only

having to evaluate a single track of animation at a time, which is not the case when using cross-fade blends.

To support different motion styles, we precompute offline the different sets of ranges in the database that correspond to each style tag, and at runtime use these ranges to limit the search domain. We find this is both more efficient in memory usage and runtime performance than adding the gait information into the feature vector using one-hot encoding.

While this simple method proves an effective way to perform kinematic character control, a naive implementation of the search is too slow for the performance constraints of games, and as such we accelerate the search using a KD-Tree, as suggested in Clavet [2016].

6 REINFORCEMENT LEARNING

In this section we describe the reinforcement learning system used to provide feedback control to the simulated character.

6.1 Simulation Environment

A simulation environment is created in C++ using the Bullet physics engine [Coumans 2015]. Maximal coordinates are used, with constraints holding the bodies together at their joints. Velocity based constraints are used to simulate PD servo motors at the joints, with motor constraint torques clamped to 200 Nm. All coefficients of friction are given a value of 1, except rolling friction which is disabled. A simulation frequency of 60 Hz is used, with 32 constraint solver iterations per simulation step.

6.2 Policy Gradient

Policy gradient methods are well suited for nonlinear control tasks in continuous state and action spaces [Sutton and Barto 2018]. We represent a policy $\pi_{\boldsymbol{\theta}}(\mathbf{s})$ with a deep neural network, parameterized by weights and biases $\boldsymbol{\theta} \in \mathbb{R}^n$, and trained to take as input states $\mathbf{s} \in \mathbb{R}^{110}$ and output actions $\mathbf{a} \in \mathbb{R}^{25}$.

During training the policy is stochastic, providing a probability of taking a specific action $\mathbf{a}$ given some state $\mathbf{s}$, denoted by $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$. In our work this is achieved by having the policy output the mean of a Gaussian distribution associated with each action, and including the standard deviations for each distribution as additional parameters in $\boldsymbol{\theta}$. A trained policy can be made deterministic by setting the standard deviations to 0.

To train the network we require a measure of performance $J(\boldsymbol{\theta})$. This can be defined as follows. First, we define a reward value r_{t+1} each time step t that results from the current state $\mathbf{s}_t$ and action taken $\mathbf{a}_t$. A value function $v_{\pi_{\boldsymbol{\theta}}}(\mathbf{s})$ is then defined, giving the expected weighted sum of rewards starting at state $\mathbf{s}$ and following the policy $\pi_{\boldsymbol{\theta}}$ afterwards until termination.

The value function is written as

$$v_{\pi_{\boldsymbol{\theta}}}(\mathbf{s}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \middle| \mathbf{s}_t = \mathbf{s}\right],$$

where $\gamma \in [0, 1]$ is the discount factor, which we set to 0.99. The performance of a particular policy can then be written in terms of the value of an entire episode starting at some start state $\mathbf{s}_0$,

$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_0).$$

Policy gradient methods approximate the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ by sampling multiple trajectories resulting from use of the policy $\pi_{\boldsymbol{\theta}}$ [Sutton and Barto 2018], and optimizing $\boldsymbol{\theta}$ iteratively using the gradient approximations in order to improve policy performance. In our work we use proximal policy optimization (PPO) [Schulman et al. 2017] due to its demonstrated success in previous work training policies for character control tasks [Chentanez et al. 2018; Heess et al. 2017; Peng et al. 2018]. We use OpenAI's "baselines" [Dhariwal et al. 2017] implementation of PPO. This implementation is freely available and mirrors the implementation described in Schulman et al. [2017]. We choose the distributed CPU version of the algorithm, with 8 simulations in parallel, and achieve an average of 2500 environment steps per second on a single machine.

6.3 States

At each control step the policy is provided with a state $\mathbf{s}_t \in \mathbb{R}^{110}$. The state is constructed as follows. First the center of mass (CM) velocity is calculated for the kinematic character $\mathbf{v}_{kin} \in \mathbb{R}^3$ and simulated character $\mathbf{v}_{sim} \in \mathbb{R}^3$. The desired horizontal CM velocity from user-input is also considered $\mathbf{v}_{des} \in \mathbb{R}^2$, and the difference between current simulated character horizontal CM velocity and $\mathbf{v}_{des}$ is calculated giving $\mathbf{v}_{diff} \in \mathbb{R}^2$. For a subset of bodies: { `LeftToe`, `RightToe`, `Spine`, `Head`, `LeftForeArm`, `RightForeArm` }, we compute positions and velocities then concatenate these giving $\mathbf{s}_{sim} \in \mathbb{R}^{36}$ for the simulated character and $\mathbf{s}_{kin} \in \mathbb{R}^{36}$ for the kinematic character. The smoothed actions produced in the previous step of the policy are collected in $\mathbf{y}_{t-1} \in \mathbb{R}^{25}$ (see Section 6.4).

Animated character body positions, velocities, CM velocity, and desired CM velocity are resolved in reference frame $\mathcal{F}_{kin}$, which is formed from the horizontal heading of the kinematic character, the gravity direction, and positioned at the character CM. The exact same procedure is used for the simulated character positions and velocities, except with a frame $\mathcal{F}_{sim}$, which is identical to $\mathcal{F}_{kin}$ in orientation but positioned at the simulated character's CM position. Note that when velocities are decomposed into $\mathcal{F}_{kin}$ or $\mathcal{F}_{sim}$, the reference frames are considered to have no angular or linear velocity and acceleration so that global velocity features are measurable in the state.

All quantities are concatenated into a single large state vector, $\mathbf{s}_t = \{\mathbf{v}_{kin}, \mathbf{v}_{sim}, \mathbf{v}_{sim}-\mathbf{v}_{kin}, \mathbf{v}_{des}, \mathbf{v}_{diff}, \mathbf{s}_{sim}, \mathbf{s}_{sim}-\mathbf{s}_{kin}, \mathbf{y}_{t-1}\} \in \mathbb{R}^{110}$. Note $\mathbf{v}_{sim} - \mathbf{v}_{kin}$ and $\mathbf{s}_{sim} - \mathbf{s}_{kin}$ are included, which allows errors to be used by the first layers of the network and improves performance, but isn't strictly necessary. Unlike Peng et al. [2018], no phase information is needed since information relating to kinematic character tracking is provided explicitly.

6.4 Actions

At each control step actions $\mathbf{a}_t$ are generated by the policy. The kinematic controller provides the initial open-loop PD control targets for the simulated character by setting the proportional control target for each DOF to the corresponding DOF joint angle output by motion matching. The feedback policy provides adjustment offsets to these targets for the following subset of joints: { `LeftUpLeg`, `LeftLeg`, `LeftFoot`, `LeftToe`, `RightUpLeg`, `RightLeg`, `RightToe`, `Spine`, `LeftArm`, `RightArm` } (see Fig 4). For 3 DOF joints, offsets are

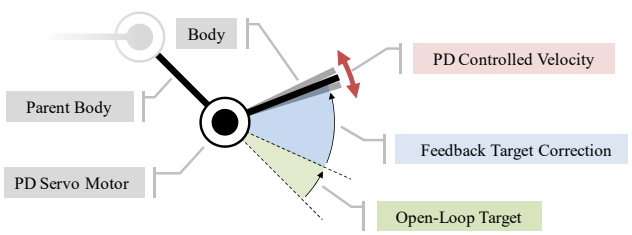

Fig. 5. For each DOF, the policy applies a feedback control correction to the usually unstable open-loop control targets. These targets are used to specify PD controlled velocities for the motors which are implemented using velocity constraints with maximum allowable torques of 200 Nm.

encoded as 3D vectors, where the magnitude provides the rotation angle and the direction gives the rotation axis. This is converted to a rotation tensor that is considered local to the bone's parent's frame on the simulated character and is used to apply a correction to the open-loop control targets by composing rotations. For 1 DOF joints (elbows, knees, toes), a single value is output by the policy and simply summed with the open-loop control target, as shown in Fig 5.

High frequency oscillation can occur if the policy input varies quickly. High-frequency control changes lead to unnatural character motion even when they improve the overall policy performance. In order to remedy this we use two strategies. First, we lower the policy evaluation rate, limiting the policy to take one action $\mathbf{a}_t$ every k simulation steps, and consider the action to remain fixed across those k steps. Second, we apply a filtering strategy to smooth the actions before they are applied to the character. A recursive exponentially weighted moving average filter is used [Ostertagova and Ostertag 2012],

$$\mathbf{y}_t = \beta\, \mathbf{a}_t + (1 - \beta)\, \mathbf{y}_{t-1},$$

where $\mathbf{y}_t$ and $\mathbf{a}_t$ are the output and input of the filter at time t respectively. Here, the action stiffness β is a hyperparameter controlling the filter strength. Lowering policy evaluation rate allows for lowered runtime costs and faster training, but also gives individual actions more weight since they are applied over multiple simulation steps. In our case we found setting $k = 2$ and $\beta = 0.2$ gave the best visual results in our experiments. This reduces behaviours that make the character look twitchy, although it can have an impact on maximum performance (see Fig 7). To ensure the filtering is visible to the policy we also provide $\mathbf{y}_{t-1}$ in the state $\mathbf{s}_t$. The quantitative effect of action stiffness on rewards can be visualized in Fig 6. An ablation study considering both the state and action representation can be found in Section 7.3.

6.5 Reward

A reward is supplied by the environment each control step that allows the quality of the action taken by the policy to be measured. Rewards are structured similarly to previous work using reinforcement learning for physically based motion capture tracking by Peng et al. [2018] and Chentanez et al. [2018]. However, we do not use standard orientation based distance metrics, and instead use alignment of multiple points $i \in [1, 6]$ per body j distributed on

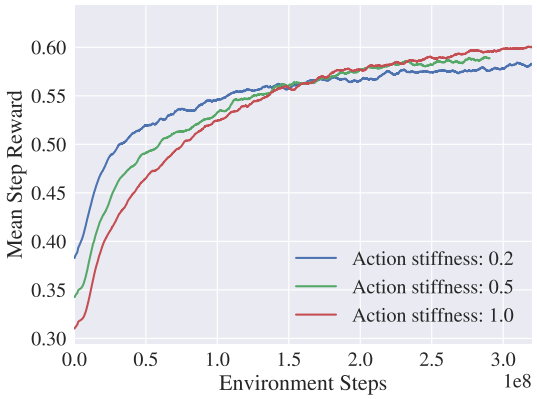

Fig. 6. Effect of action stiffness β on rewards. At the beginning of training, smooth actions help local tracking. Although rewards end up quantitatively better when stiffness is high, the visual result sometimes looks twitchy when actions change too quickly.

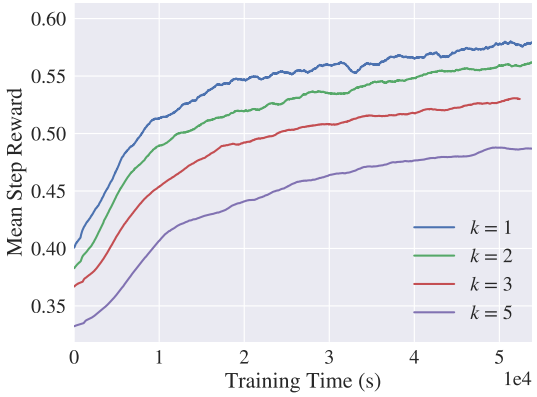

Fig. 7. Reducing the policy evaluation rate k can result in lower rewards. However, the visual results often look smoother at low frequency and motion quality is superior.

the "surface" of the geometry associated with the bodies. Surface points are fixed on the face centers of the oriented bounding box surrounding the geometry. This captures orientation and position tracking in the same reward. Maximum reward is given when the simulated character and kinematic character are perfectly synchronized. Reward falls off based on the error between the states of the two characters. If the character is detected to have fallen, the episode is terminated with zero reward. The fall is detected when the distance between the kinematic and simulated heads becomes larger than 1 m.

We break the reward up into multiple terms and weight their individual effects differently. The *position reward* is given by,

$$r_p = \exp\left(\frac{-10}{N_{bodies}} \sum_{i=1}^{6} \sum_{j=1}^{N_{bodies}} \left\| \hat{\mathbf{p}}_{ij} - \mathbf{p}_{ij} \right\|_2 \right),$$

Table 2. Comparison of our method and similar existing work on reported runtime costs.

Method	Policy Network Size	Simulation Frequency	Network Query Rate
(Ours)	2 hidden layers, 128 units each	60 Hz	30 Hz
Peng et al. [2018]	2 hidden layers, 1024 & 512 units	1200 Hz	30 Hz
Chentanez et al. [2018]	3-6 hidden layers, 512 units each	(not reported)	(not reported)

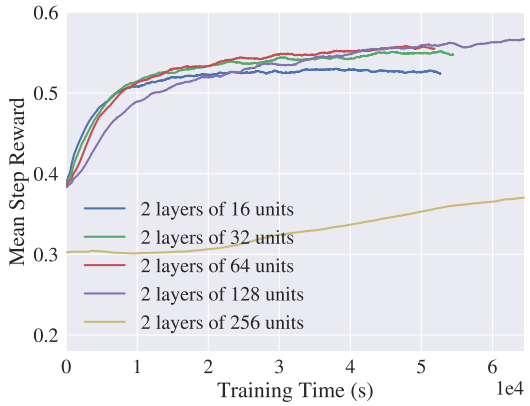

Fig. 8. Experiments with different policy network sizes. The value network used by PPO has the same size as the policy. Relatively small networks learn quickly and give acceptable policies. Our best results were achieved with only two layers of 128 neurons.

where N_{bodies} is the total number of rigid bodies on the physically simulated character and $\hat{\mathbf{p}}_{ij}$ represents the position of surface points $i \in [1, 6]$ of body j of the kinematic character, resolved in $\mathcal{F}_{kin}$. $\mathbf{p}_{ij}$ represents the position of surface points $i \in [1, 6]$ of body j of the simulated character, resolved in $\mathcal{F}_{sim}$. This element of the reward promotes horizontal local alignment of body positions between the simulated character and kinematic character, as well as global orientation alignment.

The *velocity reward* is given by,

$$r_v = \exp\left(\frac{-1}{N_{bodies}} \sum_{i=1}^{6} \sum_{j=1}^{N_{bodies}} \left\|\hat{\mathbf{v}}_{ij} - \mathbf{v}_{ij}\right\|_2\right),$$

where $\hat{\mathbf{v}}_{ij}$ represents the velocity of surface points $i \in [1, 6]$ of body j of the kinematic character in the target motion, resolved in $\mathcal{F}_{kin}$. $\mathbf{v}_{ij}$ represents the velocity of surface points $i \in [1, 6]$ of body j of the simulated character, resolved in $\mathcal{F}_{sim}$. This component of the reward promotes matching of global velocities and angular velocities between the bodies of the simulated character and the kinematic character.

The *local pose reward* is given by,

$$r_{local} = \exp\left(\frac{-10}{N_{bodies}} \sum_{j=1}^{N_{bodies}} \left\|\hat{\mathbf{a}}_j \ominus \mathbf{a}_j\right\|_q\right),$$

where $\hat{\mathbf{a}}_j$ represents the local orientation of body j of the kinematic target character expressed in its parent's frame as a quaternion. $\mathbf{a}_j$ represents the local orientation of body j of the simulated character expressed in its parent's frame as a quaternion. The magnitude of the angle of rotation $\|\cdot\|_q$ resulting from the quaternion difference, $\hat{\mathbf{a}}_j \ominus \mathbf{a}_j$, is used. This element of the reward promotes purely local pose matching without regard for global character orientation. This helps prevent the learned policy from changing the overall pose shape too much.

The *center of mass velocity reward* is given by,

$$r_{v_{CM}} = \exp\left(-\left\|\hat{\mathbf{v}}_{CM} - \mathbf{v}_{CM}\right\|_2\right),$$

where $\hat{\mathbf{v}}_{CM}$ represents the velocity of the CM of the kinematic character and $\mathbf{v}_{CM}$ represents the velocity of the CM of the simulated character, resolved in $\mathcal{F}_{kin}$ and $\mathcal{F}_{sim}$ respectively. This reward encourages matching of global movement speed.

The *fall factor*,

$$e_{fall} = \text{clamp}(1.3 - 1.4\left\|\hat{\mathbf{p}}_{CM_{head}} - \mathbf{p}_{CM_{head}}\right\|_2, 0, 1),$$

is used to modulate the final reward based on the distance between the head of the kinematic character $\hat{\mathbf{p}}_{CM_{head}}$ and the head of the simulated character $\mathbf{p}_{CM_{head}}$. This allows all rewards to decrease continuously together when the simulated character falls past a certain threshold, with rewards reaching zero before the episode is terminated.

The *reward per simulation step* is given by,

$$r_t = e_{fall}\left(r_p + r_v + r_{local} + r_{v_{CM}}\right).$$

The exponential with a negative argument is used in many terms. When used, the distances can be remapped to reach a maximum value of 1 for minimum distance, and approach 0 for large distances. Additionally, the exponential of summed negative distances is equivalent to the product of exponentials of the individual negative distances. Since these range from 0 to 1, when multiplied the largest distance term will create a modulation envelope and prevent additional reward from being received. This forces the policy to improve upon the worst tracked body to maximize reward. The optimization will distribute performance across the bones more so than if the terms were combined through simple summation.

6.6 Neural Network Design

The policy is represented as a 2-layer fully connected neural network. The input layer has 110 units, corresponding to the elements of $\mathbf{s}_t$. Each hidden layer consists of 128 hidden units. We use `tanh` activations, which empirically gives the smoothest output for small networks. The neural network has 25 output units, corresponding to the elements of $\mathbf{a}_t$. The output units represent the multivariate mean μ of a multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$. The covariance matrix has a diagonal form $\Sigma = \text{Diag}(\sigma)$, where σ is a set of parameters for the policy, optimized during training. At test time a deterministic policy can be used by setting $\sigma = 0$.

A second 2-layer 128 unit network is used to represent the value function with 110 input units, again corresponding to the elements

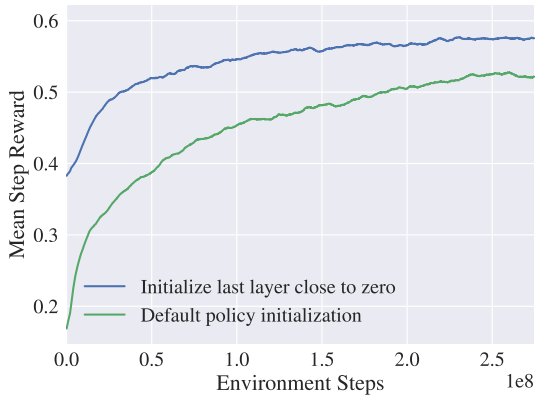

Fig. 9. Policy network initialization. Initializing smaller weights for the final layer leads to faster training since open-loop behaviour is better exploited.

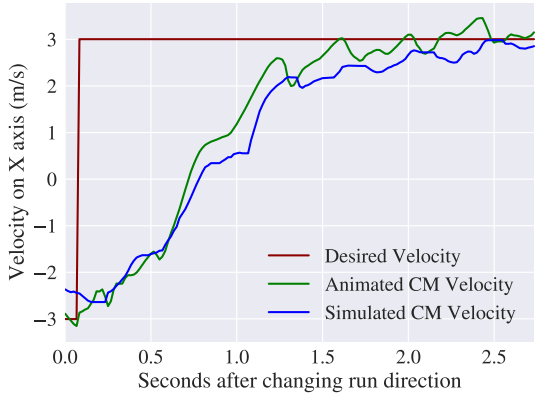

Fig. 10. A demonstration of responsiveness. A user requests a velocity change from -3 m/s to 3 m/s. The rise time is around 1.5 seconds, which is similar to the response of the kinematic motion, and thus similar to the actual time required by the motion capture actor.

of $\mathbf{s}_t$, and a single output unit to represent the estimated value function. The estimate of the value function is used by PPO. Please refer to Schulman et al. [2017] for more detail.

Care must be taken when initializing the last layer of the policy network. Standard initialization methods result in large poorly performing actions at the beginning of training, which perform worse than if the starting actions are initialized to be closer to zero such that the open-loop control dominates. For this reason we initialize the final layer weights to small values uniformly sampled between -0.01 and 0.01, which lets the open-loop control influence early training. For a visualization of the effect this has on training performance, see Fig 9.

We found that using a smaller policy network generally performed better than a larger network. For a comparison of different layer counts and sizes see Fig 8.

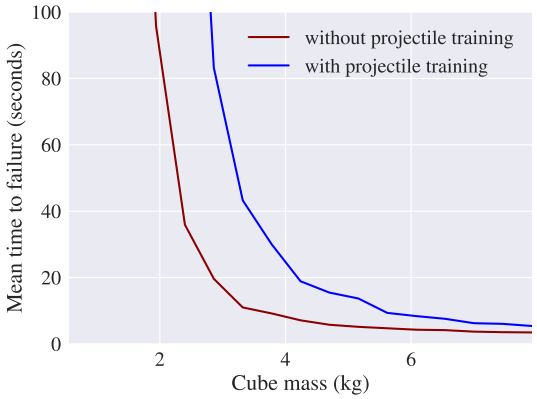

Fig. 11. Average time between falls after impact test described in Section 7.1. The character runs at 3 m/s and switches direction every 4 seconds during this test. Robustness is high even when a policy trained without projectiles is used.

6.7 Training Procedure

At the start of training, the kinematic target character is initialized in a random pose from the motion capture database. Artificial user input is continuously provided to control the kinematic character. The artificial user has a probability $P(A) = 0.001$ to make discrete control change events A each frame. These control changes consist of choosing random uniformly distributed values for: desired movement direction, desired heading direction, constant turn rate, desired movement speed, and desired movement style between walk/run/crouch. It is important that the resulting generated kinematic motion from this process captures the complete variety of possible character movements and transitions a user may cause, since this is required to maximize trained controller performance when real user input is used.

Episodes consist of finite length tracking attempts. The simulated character is initialized overlapping the current kinematic character pose. The current policy is evaluated in stochastic mode, and experience is sampled. If the character is detected to have failed by the fall check heuristic, the episode is terminated. If the episode is longer than 20 seconds it is also terminated. 40000 samples are gathered per epoch, after which PPO optimizes the policy and value networks. This is repeated until the expected return of the policy has reached a performance plateau, usually after training for around 3×10^8 steps (around 30 hours). The kinematic character is not reset when the episode terminates, and we simply reinitialize the physical character to the current kinematic character state.

7 RESULTS

In this section, we show the results of our method with respect to responsiveness and robustness, and we perform various ablation studies to evaluate our design decisions.

7.1 Robustness

We evaluate the trained controller performance through impact testing. In all impact tests, we use a 0.2 m side length cube, with a

Table 3. Response time comparison with other work. The task of commanding a simulated character to make a 90° heading change is compared. **These values were inferred from videos associated with the cited papers. The shortest response time observed for 90° heading changes are used.*

Method	Rise Time
(Ours)	1.2s
Peng et al. [2018] - *Multi-Clip Reward*	*2.0s
Liu and Hodgins [2018] - *Running Skill Graph*	*5.1s

variable mass between 0.01 kg and 8 kg. The cube is launched at 5 m/s, towards a uniformly sampled location on the vertical axis, from −0.5 m to 0.5 m, centered on the character's CM. Launches occur every second, with the cube remaining in the scene until it is relaunched (the character can trip over it after it comes to rest during this time).

Impact testing is performed by automatically controlling the character so that it runs at 3 m/s and switches direction every 4 seconds. Failure occurs when the episode ends, using same criteria used during training. The results of this test are summarized in Fig 11. Robustness is demonstrated even if the projectiles are not used during training. Training with projectiles significantly increases robustness, without visibly impacting the resulting motion style. This is because perturbed states can be better explored during training.

7.2 Responsiveness

We also evaluate the responsiveness of our control by measuring the required time for the simulated character to transition from its current motion to a new motion input by the user. The response to user input associated with a 180° direction change can be visualized in Fig 10. Note commanded directions are global, not relative. In general our simulated character response has only minor divergences from the kinematic character response. The responsiveness of more nuanced control variations such as subtle turns or transitions is best viewed from the video results, but can be inferred from Fig 2.

Both Peng et al. [2018] and Liu and Hodgins [2018] demonstrate variations of their controllers tracking heading angle changes. In Table 3, a rough comparison of best-case rise times for the task of changing heading by 90° is shown. Our method performs favorably.

7.3 State and Action Ablations

In this section we perform ablation studies for various design decisions of our method, all of which can have effects on the robustness, visual quality, and tracking quality.

The state provided to the policy must contain information regarding the simulated character state and current configuration of the generated kinematic motion in order to provide non-linear feedback control. Omitting the kinematic state results in worse motion quality, but surprisingly, a usable policy is still learned since the kinematic character state can be inferred from the open-loop control effects. We found the best results were obtained by giving the simulated character state (containing position and velocity) and the error relative to the kinematic character state (see Fig 12).

We also compared using the subset of bodies in the state and action vs. including the entire set of bodies in the state. While we would expect additional information about the complete simulated character configuration would be helpful, in practice keeping the state and action small improves results and learning speed, as shown in Fig 13. This seems to be because stochastic exploration and credit assignment becomes more straightforward with fewer actions, and redundant information is excluded from the state.

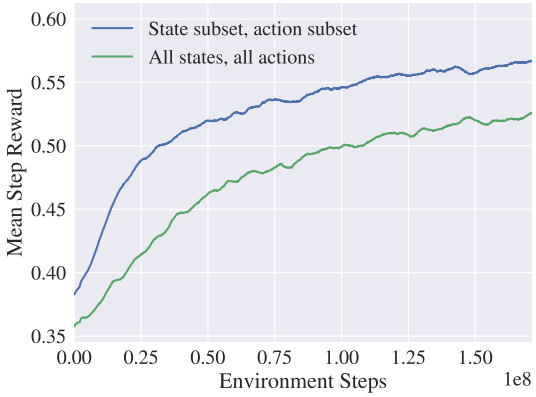

Fig. 12. Effect of using only a carefully chosen subset of bones in the state and motor offset actions. Learning is faster and achieves a better motion quality.

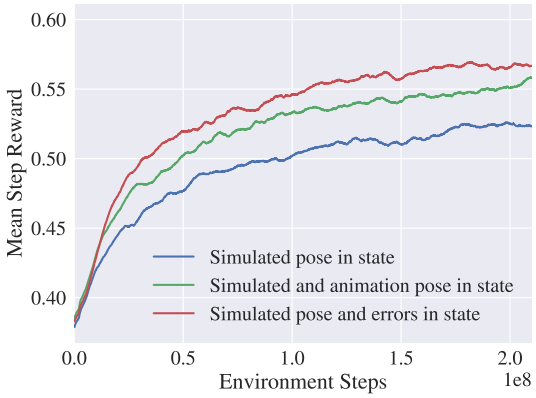

Fig. 13. State ablations. The state provided to the policy should contain sufficient information about the simulated character state and the current state of the generated kinematic animation to provide feedback control.

7.4 Reward Design Ablations

The precise details of the reward structure have small impact on learning to avoid falls, but have significant impact on the visual quality of the simulated character motion. Fig 14 describes the impact on episode lengths when we remove pieces of the reward function. It is difficult to compare quantitatively the contributions of the different parts of the reward on visual quality, refer to the video for a visualization of how the reward structure can have noticeable effects on the naturalness of the resulting character motion.

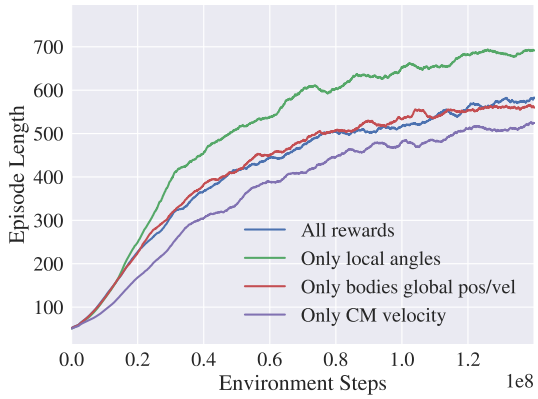

Fig. 14. Reward ablations. As this graph compares reward composition, we plot episode lengths instead of average rewards. Including only local imitation of joint angles can make learning slightly faster, but the style and global velocity tracking can degrade.

7.5 Self Collision

Self collision is usually disabled as the character geometry has realistically proportioned upper legs/spine segments which will often intersect. Self collision can be enabled without significant impact to robustness/training times, but will tend to increase the wideness of the stance as compared to the animation. Motion capture actors have soft tissues whereas our character is entirely rigid, so some compensation is automatically learned to prevent fall-prone intersections. Refer to the supplemental video for an example of this effect.

7.6 Runtime Cost

Our method has a remarkably low runtime cost, placing it within the typical budget for character animation and physics provided by a AAA video-game. This is due to the fast kinematic controller, the low simulation frequency used, and the low cost of policy evaluation. A summary of the performance of our method and the breakdown of the cost at each stage is provided by Table 4.

We compare our method to recent similar work focused on animation tracking using reinforcement learning [Chentanez et al. 2018; Peng et al. 2018]. Here, we perform a comparison in terms of policy network size, simulation frequency and policy evaluation frequency. Details are provided in Table 2. All performance measures were taken single threaded on an Intel Xeon E5-1650 V3 CPU. It should be noted that because of our lower simulation frequency some constraint solver noise is visible in the simulation. Higher frequencies can be used but lead to significant performance impacts and increases in training time since physics simulation dominates the runtime cost. Reducing the constraint solver error reduction parameter (ERP) helps to soften constraints and reduce noise [Catto 2011].

8 CONCLUSION AND FUTURE WORK

We believe our work provides a simple but powerful method of controlling simulated characters. Our data-driven strategy retains the style of natural human motion, while demonstrating a high degree of responsiveness to user input, and robustness to disturbances. We build upon recent advances that allow simulated characters to reproduce motion capture quality movement, by instead tracking natural looking generated kinematic motion that a user can easily control in real-time. The kinematic motion drives unstable open-loop control. Feedback from a control policy trained using reinforcement learning provides final stabilization adjustments. We demonstrate our final controller design maintaining character balance under a variety of challenging requirements. Our method has an impressively low runtime cost and is not dependent on high physics simulation frequencies for stability, making it well suited for real-time applications such as video-games.

Table 4. Performance breakdown of our method.

Stage	Runtime
Kinematic Control	40μs
State Construction	40μs
Policy Evaluation	60μs
Physics Simulation	200μs
Total	**340μs**

While our control method is responsive and robust, limitations remain. We focused our work on humanoid locomotion on flat terrain. While our controller can walk over slopes and obstacles without falling, it is only a consequence of the robustness of our controller. It would be much more interesting to instead see similar data-driven control methods applied towards using data captured on more complex terrain. So far our method only excels on locomotion tasks. Other movements have yet to be attempted, but we believe that our framework should naturally extend to more varied tasks if this data is supplied. Additionally, our character can sometimes fall when control requirements are too severe, and we have not yet developed good methods of making a character stand up from a fall in a natural looking way derived from a motion capture database. A solution could be to provide the simulated character state as motion matching input in order to generate an optimal get-up sequence. Another promising direction would be to modify kinematic controller input using the policy in order for it to make high-level motion planning decisions.

We hope that as research in physical character animation continues to reap benefits from advancements in the field of reinforcement learning, robust controllable simulated motion could soon become visually indistinguishable from a motion capture recording. We believe our work is a step towards that goal, and are excited to see how it can be applied in the future.

ACKNOWLEDGMENTS

We thank Ubisoft La Forge for providing funding, resources, and reference data for this project, as well as the anonymous reviewers for their helpful feedback. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and Mitacs through the Mitacs Accelerate program.

REFERENCES

David Bollo. 2016. Inertialization: High-Performance Animation Transitions in 'Gears of War'. In *Proc. of GDC 2018.*

David Bollo. 2017. High Performance Animation in Gears of War 4. In *ACM SIGGRAPH 2017 Talks (SIGGRAPH '17)*. ACM, New York, NY, USA, Article 22, 2 pages. https://doi.org/10.1145/3084363.3085069

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *CoRR* abs/1606.01540 (2016). arXiv:1606.01540 http://arxiv.org/abs/1606.01540

David F. Brown, Adriano Macchietto, KangKang Yin, and Victor Zordan. 2013. Control of Rotational Dynamics for Ground Behaviors. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '13)*. ACM, New York, NY, USA, 55–61. https://doi.org/10.1145/2485895.2485906

Michael Buttner. 2019. Machine Learning for Motion Synthesis and Character Control in Games. In *Proc. of I3D 2019.*

Erin Catto. 2011. Soft Constraints. In *Proc. of GDC 2011.*

Nuttapong Chentanez, Matthias Müller, Miles Macklin, Viktor Makoviychuk, and Stefan Jeschke. 2018. Physics-based motion capture imitation with deep reinforcement learning. 1–10. https://doi.org/10.1145/3274247.3274506

Simon Clavet. 2016. Motion Matching and The Road to Next-Gen Animation. In *Proc. of GDC 2016.*

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2010. Generalized Biped Walking Control. *ACM Trans. Graph.* 29, 4, Article 130 (July 2010), 9 pages. https://doi.org/10.1145/1778765.1781156

Erwin Coumans. 2015. Bullet Physics Simulation. In *ACM SIGGRAPH 2015 Courses (SIGGRAPH '15)*. ACM, New York, NY, USA, Article 7. https://doi.org/10.1145/2776880.2792704

Danilo da Silva, Rubens Nunes, Creto Vidal, Joaquim B. Cavalcante-Neto, Paul G. Kry, and Victor Zordan. 2017. Tunable Robustness: An Artificial Contact Strategy with Virtual Actuator Control for Balance: Tunable Robustness. *Computer Graphics Forum* 36 (03 2017). https://doi.org/10.1111/cgf.13096

Marco da Silva, Yeuhi Abe, and Jovan Popovic. 2008. Simulation of Human Motion Data using Short-Horizon Model-Predictive Control. *Computer Graphics Forum* 27 (04 2008), 371 – 380. https://doi.org/10.1111/j.1467-8659.2008.01134.x

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. 2017. OpenAI Baselines. https://github.com/openai/baselines.

Kai Ding, Libin Liu, Michiel van de Panne, and KangKang Yin. 2015. Learning Reduced-order Feedback Policies for Motion Skills. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '15)*. ACM, New York, NY, USA, 83–92. https://doi.org/10.1145/2786784.2786802

Thomas Geijtenbeek and Nicolas Pronost. 2012. Interactive Character Animation Using Simulated Physics: A State-of-the-Art Review. *Comput. Graph. Forum* 31, 8 (Dec. 2012), 2492–2515. https://doi.org/10.1111/j.1467-8659.2012.03189.x

Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. 2013. Flexible Muscle-based Locomotion for Bipedal Creatures. *ACM Trans. Graph.* 32, 6, Article 206 (Nov. 2013), 11 pages. https://doi.org/10.1145/2508363.2508399

Nikolaus Hansen. 2006. *The CMA Evolution Strategy: A Comparing Review.* Springer Berlin Heidelberg, Berlin, Heidelberg, 75–102. https://doi.org/10.1007/3-540-32494-1_4

Geoff Harrower. 2018. Real Player Motion Tech in 'EA Sports UFC 3'. In *Proc. of GDC 2018.*

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. 2017. Emergence of Locomotion Behaviours in Rich Environments. *CoRR* abs/1707.02286 (2017). arXiv:1707.02286 http://arxiv.org/abs/1707.02286

Daniel Holden. 2018. Robust Solving of Optical Motion Capture Data by Denoising. *ACM Trans. Graph.* 37, 4, Article 165 (July 2018), 12 pages. https://doi.org/10.1145/3197517.3201302

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned Neural Networks for Character Control. *ACM Trans. Graph.* 36, 4, Article 42 (July 2017), 13 pages. https://doi.org/10.1145/3072959.3073663

Sumit Jain and C. Karen Liu. 2011. Modal-space Control for Articulated Characters. *ACM Trans. Graph.* 30, 5, Article 118 (Oct. 2011), 12 pages. https://doi.org/10.1145/2019627.2019637

Andrew Kermse. 2004. Game Programming Gems 4. (2004), 95–101.

Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010. Data-driven Biped Control. *ACM Trans. Graph.* 29, 4, Article 129 (July 2010), 8 pages. https://doi.org/10.1145/1778765.1781155

Yoonsang Lee, Moon Seok Park, Taesoo Kwon, and Jehee Lee. 2014. Locomotion Control for Many-muscle Humanoids. *ACM Trans. Graph.* 33, 6, Article 218 (Nov. 2014), 11 pages. https://doi.org/10.1145/2661229.2661233

Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. 2018. GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning. *CoRR* abs/1810.05762 (2018). arXiv:1810.05762 http://arxiv.org/abs/1810.05762

Libin Liu and Jessica K. Hodgins. 2017. Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning. *ACM Transactions on Graphics* 36, 3 (2017).

Libin Liu and Jessica K. Hodgins. 2018. Learning Basketball Dribbling Skills Using Trajectory Optimization and Deep Reinforcement Learning. *ACM Trans. Graph.* 37, 4, Article 142 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201315

Libin Liu, Michiel van de Panne, and Kangkang Yin. 2016. Guided Learning of Control Graphs for Physics-Based Characters. *ACM Trans. Graph.* 35, 3, Article 29 (May 2016), 14 pages. https://doi.org/10.1145/2893476

Libin Liu, KangKang Yin, and Baining Guo. 2015. Improving Sampling-based Motion Control. *Comput. Graph. Forum* 34, 2 (May 2015), 415–423. https://doi.org/10.1111/cgf.12571

Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. 2010. Sampling-based Contact-rich Motion Control. *ACM Transctions on Graphics* 29, 4 (2010), Article 128.

Adriano Macchietto, Victor Zordan, and Christian R. Shelton. 2009. Momentum Control for Balance. *ACM Trans. Graph.* 28, 3, Article 80 (July 2009), 8 pages. https://doi.org/10.1145/1531326.1531386

Uldarico Muico, Yongjoon Lee, Jovan Popović, and Zoran Popović. 2009. Contact-aware Nonlinear Control of Dynamic Characters. In *ACM SIGGRAPH 2009 Papers (SIGGRAPH '09)*. ACM, New York, NY, USA, Article 81, 9 pages. https://doi.org/10.1145/1576246.1531387

Eva Ostertagova and Oskar Ostertag. 2012. Forecasting Using Simple Exponential Smoothing Method. *Acta Electrotechnica et Informatica* 12 (12 2012), 62–66. https://doi.org/10.2478/v10198-012-0034-2

Soohwan Park, Hoseok Ryu, Sunmin Lee, and Jehee Lee. 2019. Learning predict-and-simulate policies from unorganized human motion data. *ACM Trans. Graph.* 38, 6, Article 205 (Nov. 2019).

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. DeepMimic: Example-guided Deep Reinforcement Learning of Physics-based Character Skills. *ACM Trans. Graph.* 37, 4, Article 143 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201311

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2015. Dynamic Terrain Traversal Skills Using Reinforcement Learning. *ACM Trans. Graph.* 34, 4, Article 80 (July 2015), 11 pages. https://doi.org/10.1145/2766910

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2016. Terrain-adaptive Locomotion Skills Using Deep Reinforcement Learning. *ACM Trans. Graph.* 35, 4, Article 81 (July 2016), 12 pages. https://doi.org/10.1145/2897824.2925881

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel van de Panne. 2017. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)* 36, 4 (2017).

Marc H. Raibert and Jessica K. Hodgins. 1991. Animation of Dynamic Legged Locomotion. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*. ACM, New York, NY, USA, 349–358. https://doi.org/10.1145/122718.122755

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

Kwang Won Sok, Manmyung Kim, and Jehee Lee. 2007. Simulating Biped Behaviors from Human Motion Data. *ACM Trans. Graph.* 26, 3, Article 107 (July 2007). https://doi.org/10.1145/1276377.1276511

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* MIT Press.

Yuval Tassa, Tom Erez, and Emo Todorov. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 4906–4913. https://doi.org/10.1109/IROS.2012.6386025

Vicon. 2018. Vicon Software. https://www.vicon.com/products/software/

KangKang Yin, Kevin Loken, and Michiel van de Panne. 2007. SIMBICON: Simple Biped Locomotion Control. *ACM Trans. Graph.* 26, 3, Article 105 (July 2007). https://doi.org/10.1145/1276377.1276509

Fabio Zinno. 2019. ML Tutorial Day: From Motion Matching to Motion Synthesis, and All the Hurdles In Between. In *Proc. of GDC 2019.*

Victor Zordan and Jessica K. Hodgins. 2002. Motion Capture-driven Simulations That Hit and React. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '02)*. ACM, New York, NY, USA, 89–96. https://doi.org/10.1145/545261.545276